Available online at www.sciencedirect.com
ScienceDirect
Forensic Science International 176 (2008) 124–136
Forensic Science International
www.elsevier.com/locate/forsciint

Reconstruction of ripped-up documents using fragment stack analysis procedures

Patrick De Smet [a,b,1,*]

[a] Nationaal Instituut voor Criminalistiek en Criminologie(NICC/INCC), Vilvoordsesteenweg 100, B-1120 Brussel, Belgium
[b] Ghent University, Dep. Telecommunications and Information Processing, Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium

Received 12 March 2007; accepted 26 July 2007
Available online 14 September 2007

Abstract

Reconstruction of ripped-up documents can be a very time-consuming task for forensic experts. Currently, this task is often carried out using various homespun or loosely defined procedures. Under the main assumption that a (partially) ordered set of fragments can be recovered, we propose and discuss a more formal analysis methodology for this type of reconstruction problem. We discuss several complications that can occur in real-life problems and illustrate the efficiency of the proposed methods. Although several avenues for further research remain, we show that the proposed approach offers both a better understanding of the problem, as well as important strategies for devising very fast manual and semi-automatic fragment reassembly procedures.

Keywords: Ripped-up documents; Reconstruction; Stack analysis

1. Introduction

An important problem in forensic document examination is the reconstruction of ripped-up and shredded documents. Although some progress has been made in devising semi-automatic methods for reducing the mathematical complexity of reconstruction and reassembly problems using digital image scans of the remnants, many issues still remain difficult or unresolved. Hence, in practice, forensic examiners most often still resort to manual reconstruction procedures or seek expensive assistance from third parties whom, until now, have made little scientific information available about their methods.

In this paper, we discuss a formal analysis of the problem of reconstructing ripped-up documents when the remnants can be recovered as an ordered stack of fragments. We will show that the techniques discussed below can yield efficient search and reassembly procedures for both manual and semi-automatic reconstruction tasks.

The next section briefly reviews the existing literature that is related to this paper. Section 3 discusses a small set of initial conditions that we assume to be upheld throughout the remainder of this paper. The following sections discuss a typical ripping process used when destroying documents and the proposed reconstruction methods that reverse the ripping procedure. Section 6 discusses the speed-up that these basic algorithms can offer. In Sections 7–9, we discuss several real-life complications and their solutions. Section 10 summarizes and concludes this paper.

2. Existing work

To our knowledge, little or no work similar to the contents of this paper has been formally documented in the existing literature. Additionally, two open calls for collaboration and feedback while reporting some of our earlier work in this area, see refs. [1,2], have returned little additional information.

Hence, as far as we know, the only other areas of related work can be found in research efforts that are aimed at devising (semi-)automatic methods for reconstructing ripped-up docu-

* Tel.: +32 2 240 05 03/00.
E-mail addresses: Patrick.DeSmet@just.fgov.be, pds@telin.ugent.be.
[1] The author is currently employed by the Belgian National Institute of Forensic Science (NICC/INCC), but still holds an official voluntary postdoctoral research position at Ghent University. This paper is a result of research efforts carried out at both institutions mentioned above.

0379-0738/$ – see front matter © 2007 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/j.forsciint.2007.07.013

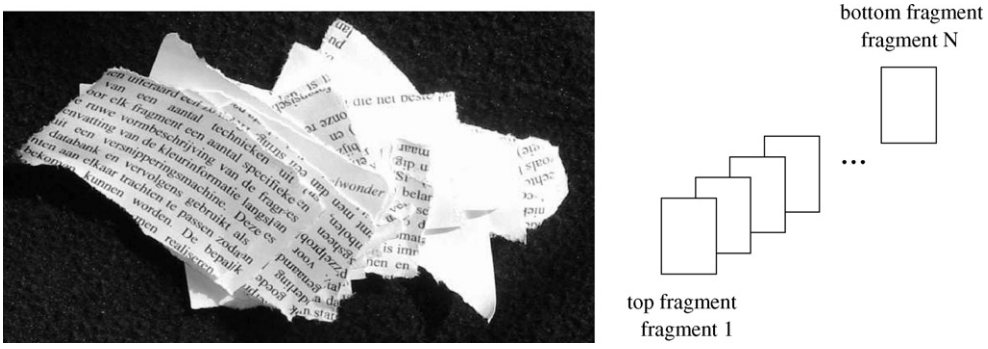

Fig. 1. An example stack of recovered fragments and its formal representation as used throughout this paper (i.e., the N fragments are numbered top-to-bottom).

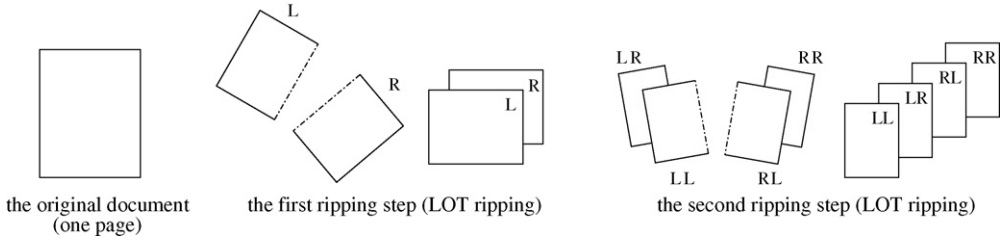

Fig. 2. An example ripping sequence using "leftmost-on-top" (LOT) fragment positioning.

ments. However, these methods are still rather inefficient and/or limited in their capabilities; we will briefly summarize and discuss some of this work.

First of all, several automated methods for solving related problems can be found in computer vision and artificial intelligence techniques for solving jigsaw puzzles; see refs. [3–6] and references mentioned therein. However, the majority of this research has been based on rather specific shape and color features, as well as the relationships that may exist between several jigsaw puzzle pieces. Some of our own work in this area has been reported in ref. [6].

For document and 2D object reconstruction problems some research has been reported in refs. [7–9]. In ref. [7], a framework for reconstructing broken tiles has been proposed. However, the reported claim that this technique would be useful for matching a very large number of fragments has not been fully demonstrated. In ref. [8], Justino et al. discuss the use of image processing techniques for automatically reassembling documents. However, the methods they report on only seem to work for very small sets of fragments (10–15) and the authors report that the performance of their techniques degrades rather rapidly. Finally, some interesting results have been reported in ref. [9]. However, this paper focuses on the reconstruction of archaeological wall-paintings. Our first results obtained for automated forensic reassembly of documents and objects have been reported in refs. [10–12]. This research is currently still being improved upon.

Hence, in our opinion many possibilities for further research remain. More specifically the design of interactive graphical user interface (GUI) tools that can be used for semi-automatic document reconstruction is in dear need of attention; all of the above references use a "one go, straight-line" processing approach, while it is clear that interactive correction and partial reinitialization of the algorithms would be required to develop practically useful tools. Additionally, more efficient and intelligent fragment matching algorithms need to be designed, implemented and experimented with.

This paper proposes several such intelligent algorithms, i.e., our methods try to fully exploit the a priori information that may be available when a ripped-up document is recovered, rather than performing inefficient semi-structured[2] or random searching and matching procedures.

3. Limitations and assumptions

The first and most important assumption that we use throughout this paper is that the remnants for which we will be trying to reconstruct the original document, will be recovered as a partially ordered set of fragments; see Fig. 1. This means that, e.g., the on-site recovery of the fragments must guarantee that the fragments are most carefully picked up, stored, transported, etc. Put into other words, the relative stack position of each fragment with respect to all other fragments must be available and retained.

Next, we assume that each fragment pairing decision that is reached will be correct, i.e., we assume that no mistakes can or will be made when determining if fragments can or can not be paired or matched to fit alongside each other.

For some of the more complicated situations that will be discussed below, additional assumptions will be introduced as soon as they are needed.

[2] A typical example of semi-structured random search is, e.g., first reconstructing the page borders (outer frame fragments can be matched/aligned more easily), followed by an incremental random "frame filling" process. Such strategies are and have already been used in both real-life as well as automated jigsaw puzzle solving methods.

4. An iterative algorithm for reconstruction of an ideal fragment stack

4.1. A simple ripping-up process model

Before we build-up a progressively more complex ripping and reconstruction model, we first discuss the ideal case of a ''single page, leftmost-on-top'' strategy.

One of the most common methods for ripping up a single page of paper consists of tearing the page in half, positioning both halves on top of each other and re-iterating this procedure as shown in Fig. 2. This figure also illustrates the ''leftmost-on-top'' principle, i.e., after each ripping step *i*, the leftmost half of the stack of fragments is positioned on top of the rightmost stack of fragments. As a result, we can denote a ripping process as a sequence of characters. Each such sequence represents the history of the information to which half the fragments belonged to in the previous ripping step. The character strings can be build-up using a prefix notation. Hence, for a single page and two ripping steps with ''leftmost-on-top'' (LOT) strategy, the resulting sequence will be: {*LL*, *LR*, *RL*, *RR*}; see also Figs. 1 and 2. For a ''rightmost-on-top'' (ROT) strategy two ripping steps will yield a ripping sequence {*RR*, *RL*, *LR*, *LL*}, and three ripping steps will yield the sequence {*RRR*, *RRL*, *RLR*, *RLL*, *LRR*, *LRL*, *LLR*, *LLL*}.

4.2. A simple reconstruction process model

For the example shown in Fig. 2, the set of fragments that would be recovered will have to be labeled 1,2,3,4, because during any reconstruction process we can not assume to have any knowledge other than the fact that we are trying to reconstruct a stack of fragments under the assumptions given above. Now, to solve the reconstruction problem for this simple example we can use two reconstruction steps (for reversing the ripping steps):

- *Reconstruction step 1*: First we try to pair fragments 1 and 3. These two fragments will match. Hence, we build a new meta-fragment labeled (1,3). Then, we match fragments 2 and 4, yielding meta-fragment (2,4).
- *Reconstruction step 2*: We now pair meta-fragment (1,3) with meta-fragment (2,4), yielding the end result, i.e., a single fragment.

Thus, instead of following the sequence of ripping operations from left to right in Fig. 2, we now reverse the process by going from right to left in Fig. 2.

Obviously, this is a trivial example, and the techniques discussed in this paper are not aimed a solving such small scale problems for which fully manual ''random'' search will often be sufficiently effective.

The main point that we wish to make here is that the proposed procedure can be easily extended to any number $N = 2^n$, $n \in \mathbb{N}_0$ and $n < \mathcal{N}$, of initial fragments, i.e., for reconstructing any ideal LOT stack of fragments we can always use the procedure summarized in Algorithm 1. The additional constraint, i.e., $n < \mathcal{N}$, indicates that the maximum number of fragments is limited, e.g., by the physical size of the fragments, by the overall thickness of the stack of fragments, or by a lack of sufficient ''fragment feature length'' for describing their shape, contour colors, etc., that is required for enabling reliable fragment pairing.

Algorithm 1. The basic iterative reconstruction algorithm

1: set reconstruction step count $c = 0$;
2: $S^{(0)}$ is the recovered stack of fragments,
3: $S^{(0)}(1)$ is the fragment on top,
4: $S^{(0)}(N)$ is the bottom fragment, set $n = \log_2 N$
5: **if** (number of fragments N is not a power of 2) **then**
6: error; stop;
7: **end if**
8: **if** $(c == n)$ **then**
9: stop; // reconstruction completed
10: **else**
11: split the current stack $S^{(c)}$ in half, yielding $S_1^{(c)}$ and $S_2^{(c)}$,
i.e., $S_1^{(c)}(i) = S^{(c)}(i)$ and $S_2^{(c)}(i) = S^{(c)}(i + N/2)$, for $i = 1, \ldots, N/2$.
12: **for all** remnants $i \in [1, \ldots, N/2]$ in $S_1^{(c)}$ **do**
13: **if** remnant $S_1^{(c)}(i)$ pairs with $S_2^{(c)}(i)$ **then**
14: form fragment $S^{(c+1)}(i) = (S_1^{(c)}(i),\ S_2^{(c)}(i))$
15: **else**
16: error; stop;
17: **end if**
18: **end for**
19: set $c = c + 1$, set $N = N/2$, go to line 8
20: **end if**

Some of the error conditions and the limitations included in Algorithm 1 will be discussed further below.

Note also that this algorithm works because it progressively reduces the ripping sequence, e.g., for the simple example given above, the algorithm pairs fragment 1, i.e., LL, with fragment 3, i.e., RL. Formulated more generally, this pairing experiment will be successful due to:

The general pairing rule:

Two fragments will match if their first ripping sequence characters are each others dual, and their suffixes are identical; a new fragment formed by joining the two matching fragments will retain the ripping sequence corresponding to the common suffix.

For our simple example this is confirmed as follows: **L**L matches **R**L yielding fragment L, and fragment 2, i.e., LR, matches fragment 4, i.e., RR, yielding fragment R.

5. An incremental build-up algorithm for reconstruction of an ideal fragment stack

An alternative method for reconstructing a document can be derived from the algorithm discussed above. Let us first illustrate this with an example.

Consider eight fragments being recovered. The algorithm formulated above would generate the following reconstruction sequence: step 0: {1,2,3,4,5,6,7,8}, step 1: {(1,5), (2,6), (3,7), (4,8)}, step 2: {(1,5,3,7), (2,6,4,8)}, step 3: {(1,5,3,7,2,6,4,8)}.

Clearly, the last sequence could be considered to be used immediately for incrementally constructing the end result. More specifically, we could use the following fragment pairing operations: step 1: (1,5), step 2: ([1,5], 3), step 3: ([1,5,3], 7), etc.

In order to generate the sequence of steps needed to perform this incremental build-up reconstruction, one can use a pen-and-paper method or simply realize a computer based implementation of Algorithm 1 in which $S^{(0)}$ is a character sequence filled with the first N natural numbers, $1, \ldots, N$, and $S^{(c)}$ will be filled-in with subsequences as illustrated by the example and the algorithmic rules given earlier in this section.

A disadvantage of this algorithm is that it requires fitting a small(er) fragment to an increasingly bigger fragment, which can require more time to actual perform the pairing step. This is true especially when (i) the fragments contain only white space, or (ii) if they are small and/or (iii) if their contours show little curvature or other features that can be matched easily.

Additionally, the algorithm needs to be adapted and (partially) recomputed whenever real-life complications are met that deviate from the ideal fragment stack situations discussed above; some of these complications are discussed below.

This may make the use of the algorithm rather difficult. Hence, the remainder of this paper is mainly focused on (adaptations of) the iterative algorithm formulated above.

6. Impact on the number of fragment matching operations

6.1. Random search methods

As already mentioned above, small scale reconstruction problems can often be solved quickly using a ''random search'' procedure for matching fragments. However, the human visual system in combination with our relatively small amount of short-term memory, is rather limited in its capabilities. Hence, as the number of fragments increases the performance of a non-structured search procedure will quickly degenerate. Let us explain this in detail.

The first fragment pairing iteration of a random search procedure will on average need $(N-1)/2$ matching operations to find a first match between two fragments. Additionally, note

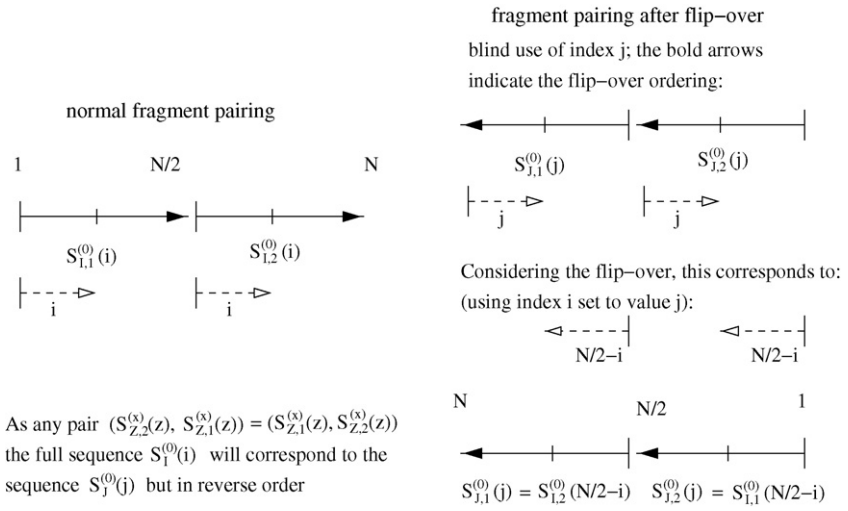

Fig. 3. Full flip-over of a stack of fragments does not require any changes to the iterative reconstruction algorithm. This figure illustrates the first reconstruction iteration with or without flip-over; each pair of fragments that needs to be formed will still be formed correctly. The iterative algorithm will either generate $S^{(1)}_{I}(i)$ or its reverse equivalent, i.e., $S^{(1)}_{J}(j)$. Hence, upon arriving at the final reconstruction iteration, only two meta-fragments will remain either in the order (a,b) or in the order (b,a). These meta-fragments will always be merged into an identical end result.

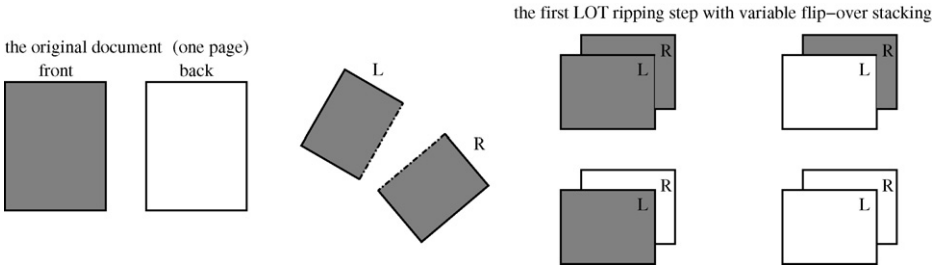

Fig. 4. A single LOT ripping sequence will result in four different fragment positioning possibilities when variable fragment flip-over stacking is considered (white colored areas indicate flipped-over fragments).

that due to the random nature of the matching process, the probability of a successful outcome of a single pairing experiment is low.

If the first match is completed $N-1$ fragments will remain to be paired further. The second match will on average require $(N-2)/2$ matching experiments after which $N-2$ fragments will remain to be matched further. This process continues until the last step when only two fragments remain and only an actual fitting step needs to be carried out. Hence, the overall mathematical complexity is of the order N^2. Basically, this complexity is the main reason why image processing techniques have been considered for automating reconstruction problems (see Section 1).

6.2. The proposed algorithms

The first iteration of the algorithm proposed above needs $N/2$ pairing, i.e., relative fragment fitting operations to reduce the remaining number of fragments to $N/2$. Additionally, after the first successful fragment pairing step, the probability of the subsequent fragment fitting operations increases rapidly in the ideal case because the hypothesis of the fragment ripping model can be assumed to be true. If we now consider the iterative pairing process as a mathematical series with factor $1/2$, it is clear that the total number of fragment matching operations for reconstructing the entire document will be N (mathematical series with first element N and factor $1/2$ converge to N).

This result is also confirmed by simply determining the number of fragment matches needed by the incremental build-up algorithm (see Section 5). As we incrementally build-up the document of N fragments, we will need exactly N steps to complete the process.

If we contrast this result with the random search procedure (see above), it is clear that the proposed algorithms can yield very important speed-ups; i.e., for N fragments the proposed matching algorithm is N times faster (on average).

7. Solutions for real-life complications: flip-overs and variable substack positioning

7.1. Full stack flip-over

In the previous paragraphs, we discussed a somewhat idealized situation. In real-life situations various events can introduce several complications.

A first problem that may occur is the (accidental) up-side-down orientation or a full flip-over of the stack, i.e., should a recovered stack be examined in the position as found on-site, or should it be flipped-over? As it turns out, both algorithms formulated above do not need to be changed to compensate for this problem. For the iterative algorithm this can be shown as follows.

Assume that we start out with $N = 2^n$ recovered fragments. If we use the standard procedure described above we will start by forming matched fragment pairs $S_I^{(1)}(i) = (S_{I,1}^{(0)}(i), S_{I,2}^{(0)}(i))$, for $i = 1, \ldots, N/2$. If we flip-over the entire stack, or conversely if the stack was already flipped-over before it could be recovered, the first iteration of Algorithm 1 will result in couples $S_J^{(1)}(j) = (S_{J,1}^{(0)}(j), S_{J,2}^{(0)}(j))$ being formed, for $j = 1, \ldots, N/2$. As Fig. 3 illustrates, this last set will correspond to the set $S_I^{(1)}(i)$, but in reversed order, i.e., $S_J^{(1)}(j) = S_I^{(1)}(N/2 - i)$ for $i = j, j = 1, \ldots, N/2$. Due to the iterative nature of the reconstruction algorithm, and the fact that all (relative) ordering relationships are always retained, the final reconstruction result will always be identical.

For the incremental build-up algorithm a similar proof can be formulated. This is quite obvious as the incremental build-up algorithm was derived directly from the iterative algorithm.

For a single sided document it can sometimes be possible to determine the full flip-over condition immediately; if no text or other document features are visible for the initial orientation, the stack will probably need to be flipped-over. For double sided documents a simple examination may not always yield immediate certainty about the preferred orientation[3]; as the number of fragments N increases the number of fragments with little or no reliably matchable features will also increase (due to white space, page margins, etc.). For such fragments one can not easily determine if they should be examined flipped-over or not.

7.2. Random substack positioning

Secondly, what would happen if we would not adhere to the LOT convention? Looking first at the situation in which the

[3] The preferred orientation is the orientation that will make the speed of the overall matching process as high as possible. Obviously, this may not be clear early on in the reconstruction process as the quality, or the "matchability", of certain meta-fragments may only become clear later on.

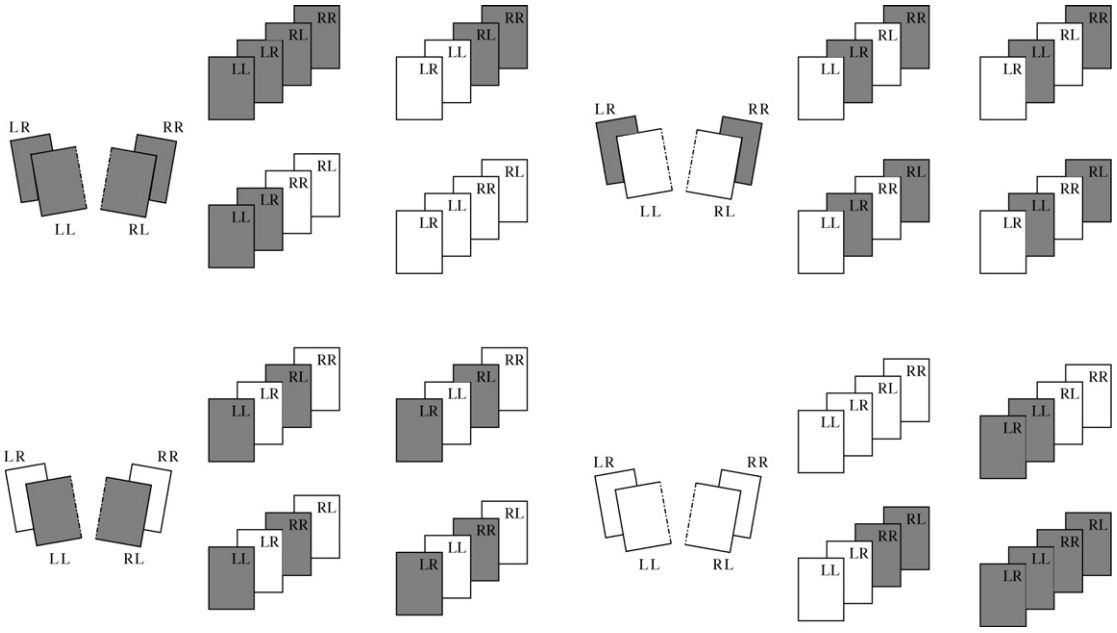

Fig. 5. Two LOT ripping sequence will result in 16 different fragment positioning possibilities when variable fragment flip-over stacking is considered.

LOT strategy would be replaced by the ROT strategy during the entire ripping process, one can easily verify that this does not make any difference with respect to the reconstruction process. The only difference is that for each reconstruction iteration R/L-pairings will be carried out instead of L/R-matches as used during a LOT-reconstruction.

Next, we can consider a ripping process in which any random order of subsequent substack positioning is used, i.e., after any ripping step either the leftmost or rightmost substack is put on top of the other. Again, this will not make any difference during the reconstruction process because at each reconstruction step the corresponding left and right fragments of each ripping step will always be paired together.

To illustrate this let us again return to the example discussed above using only two ripping steps. When we consider all possible substack positioning manipulations during the ripping steps we can obtain the following stacks: {*LL*,*LR*,*RL*,*RR*} (i.e., the result of repeated LOT ripping), {*RL*,*RR*,*LL*,*LR*} (LOT, followed by ROT), {*LR*,*LL*,*RR*,*RL*} (ROT, followed by LOT) and {*RR*,*RL*,*LR*,*LL*} (i.e., ROT, ROT ripping sequence). One can easily verify that in each case, the proposed algorithms will correctly reassemble all the fragments.

Obviously, this is also due to the fact that the general fragment pairing rule as formulated above can still be applied; all fragment pairs in both halves of the ripping sequence will always start with each others'dual letter.

7.3. Random substack flip-over

Finally, we consider complications due to ''variable flip-over'', i.e., for each ripping iteration one may decide to flip-over the fragments before stacking them. Fig. 4 illustrates all different situations that can occur when considering only a single ripping step. Fig. 5 illustrates all possibilities for two ripping steps.

The basic algorithm can not be used to reconstruct this type of fragment stacks. This is due to the fact that two fragments will not match if one of them is flipped-over; Fig. 6 illustrates this further.

Fortunately, the problem can be solved easily by deleting lines 12–18 in Algorithm 1 and replacing them by the procedure summarized in Algorithm 2.

Algorithm 2. Adaptations required for variable flip-over substack positioning

```
if S_1^(c)(1) pairs with S_2^(c)(1) then
  for all remnants i ∈ [1,...,N/2] in S_1^(c) do
    if remnant S_1^(c)(i) pairs with S_2^(c)(i) then
      form S^(c+1)(i) = (S_1^(c)(i), S_2^(c)(i))
    else
      error; stop;
    end if
  end for
else
  for all remnants i ∈ [1,...,N/2] in S_1^(c) do
    if remnant F( S_1^(c)(i) ) pairs with S_2^(c)(N+1−i) then
      form S^(c+1)(i) = ( F( S_1^(c)(i) ) , S_2^(c)(N+1−i) )
    else
      error; stop;
    end if
  end for
end if
```

The function F is used to indicate that the fragment(s) need to be flipped-over before they are assembled. Instead of flipping-over $S_1^{(c)}(i)$, as indicated in this new procedure, one can also decide to flip-over $S_2^{(c)}(N+1-i)$, i.e., one can form $S^{(c+1)}(i) = (S_1^{(c)}(i), F(S_2^{(c)}(N+1-i)))$. The decision of which

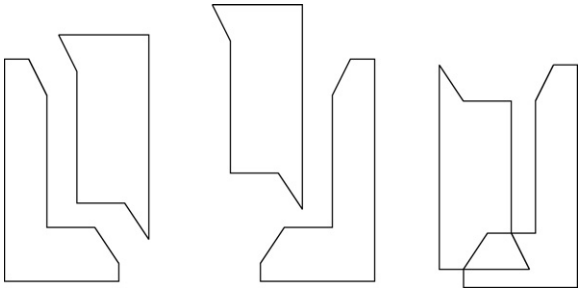

Fig. 6. Left: Example of two matching artificial fragments; right: if one of the fragments is flipped-over, i.e., the L-shaped fragment, the fragments can no longer be correctly paired and reassembled.

set of fragments to flip may depend e.g. on the fragment features that are available; e.g., for single sided documents it is normally easier to pair the segments with the text facing upwards. Please note that the choice of which $S^{(c+1)}(i)$ flip-over solution will be used, should be decided before the pairing iterations are started, i.e., generally speaking one may not change the flip-over decisions during a certain reconstruction iteration c.

Finally, it is important to note that variable flip-over can also have a severe impact on any loosely defined or random search procedure. This is especially true for double sided documents. Basically, this is due to the fact that when using random search while also considering flip-over, one has to match the first fragment with, on average, $2(N-1)/2 = N-1$ fragments (see also Section 6.1) as each fragment may also have to be considered in a flipped-over position. After this first step $N-1$ fragments still remain to be matched further, etc. For the algorithm we propose, variable flip-over will not increase the total number of required matching operations by more than N_F, with N_F equal to the total number of flip-overs that occurred during the ripping-up process. Hence, it is clear that the proposed algorithm yields another important advantage; i.e., the proposed method is (on average) approximately $2N$ times faster (with N_F unknown, worst case speed-up is $\approx 2N^2/(N+N_{F,\max}) = 2N/(1+N_{F,\max}/N)$ with $N_{F,\max} = \log_2 N$; note that due to the log operator $N_{F,\max}$ will be relatively small).

8. Reconstructing multi-page stacks

An important extension that obviously also needs to be studied is the simultaneous ripping up of multiple pages. In this case we assume the initial stack $S^{(0)}$ to contain P pages. Hence, if a traditional ripping process is used, the number of fragments will be $N = 2^n P$, for n sequential ripping iterations.

Hence, line 5 and following in Algorithm 1 need to be replaced by:

if (number of fragments N is not equal to $2^n P$) for some n and P **then**
  error; stop;
end if

No further changes are required; the algorithms formulated above can still be used.

A quick method for determining the number of pages P can be based on Table 1 (still assuming that no fragments have been lost and that no ''foreign'' fragments have been added). As can be noted, the number of ripping iterations and hence the number of fragments has been restricted. This is due to the fact that as the number of pages or ripping iterations increases, the number of possible ripping iterations will decrease due the maximum physical thickness of a stack of fragments that a person will normally rip-up. Additionally, Table 1 only considers situations for an odd number of pages. This is due to the fact that any even number of pages will reoccur in one of the other columns, e.g., the situation in which two pages are being ripped-up can be considered as a special case of the ripping-up sequence of a single page; ripping up two pages will also result in 4, 8, 16, etc., fragments, see the column for $P = 1$ in Table 1. The only difference will be the number of iterations c that will be needed to complete the reconstruction process.

Under the assumption that only traditional rectangular pages are being considered, a fast indication for determining the number of pages can also be found in counting the number of fragments with real straight-angle corners. This can be done for both manual and automatic reconstruction techniques (e.g., see ref. [12] for the image processing methods used for detecting corners). The number of corners should normally be equal to $4P$. Other indications that may help to determine the number of pages is the possible physical differences in paper quality or color, or, especially for automated methods, the total amount of straight line page borders. For a fixed paper size (A4, U.S. Letter, etc.) the total length of the page circumference is predetermined (see ref. [13]).

An interesting strategy for reconstructing a multi-page document, consists of first separating all fragments that will belong to the same page. This can be done easily: the initial stack $S^{(0)}(i), i = 1,\ldots,N$, simply needs to separated in P different stacks $S^{(0)}(p;i) = S^{(0)}((i-1)P+p)$, for $i = 1,\ldots,M$ with $M = N/P$, and $p = 1,\ldots,P$. After that, the iterative algorithm can be used (sequentially or in parallel) to process each stack $S^{(0)}(p;i)$ separately.

Unfortunately, such a procedure is not robust against other complications such as, e.g., variable flip-over or missing fragments (see below). This is why we prefer to continue adapting the iterative algorithm.

Finally, as already discussed above, for an increasing number of pages the number of possible ripping iterations will decrease, e.g., due the maximum physical thickness of a stack of fragments that a person can rip-up. However, this does not mean that performing a manual reconstruction procedure will become easier: the total size of the required desk surface will increase rapidly, making the reconstruction considerably more

Table 1
The number of fragments that will result from multiple pages being ripped-up multiple times

No. of ripping iter. (n)	No. of pages (P)										
	1	3	5	7	9	11	13	15	17	19	...
1	2	6	10	14	18	22	26	30	34	38	...
2	4	12	20	28	36	44	52	60	68	76	...
3	8	24	40	56	72	88	104	120	136	152	...
4	16	48	80	112	144	176					...
5	32	96	160								...
6	64	192									...
7	128										...

We assume that the recovery of a stack that contains more than 200 fragments is highly unlikely; numbers for P even are not shown (see also text).

difficult. For automated reconstruction methods the images and their derived numerical feature storage and computation, the digital image canvas, etc., will also grow rapidly.

Hence, the methods proposed in this paper are particularly useful for reconstructing multi-page ripped-up documents; matching fragments can be identified easily.

9. Compensating for missing fragments

One of the most important complications met in real-life situations is the loss or irreversible damage of a subset of the fragments.

Obviously, general random loss (and/or shuffling) of certain fragments may result in semi-structured or random searching to perform better or may even render the algorithms proposed in this paper inoperable. However, as already pointed out above, this paper discusses only situations in which a relatively strong ordering degree for most of the fragments has been retained. Studying more complicated missing fragment situations offers interesting avenues for further research.

As such, Fig. 7 illustrates the assumptions that we make in this section; i.e., we assume that only a single block or substack of fragments has been lost or severely damaged before the reconstruction process is started. For situations with severely damaged fragments (e.g., charcoaled fragments that do not allow any reliable matching), we assume that this block of fragments will have been removed from the stack.

As a small sidestep, please note that the algorithms proposed in this paper may also be very useful for reconstructing

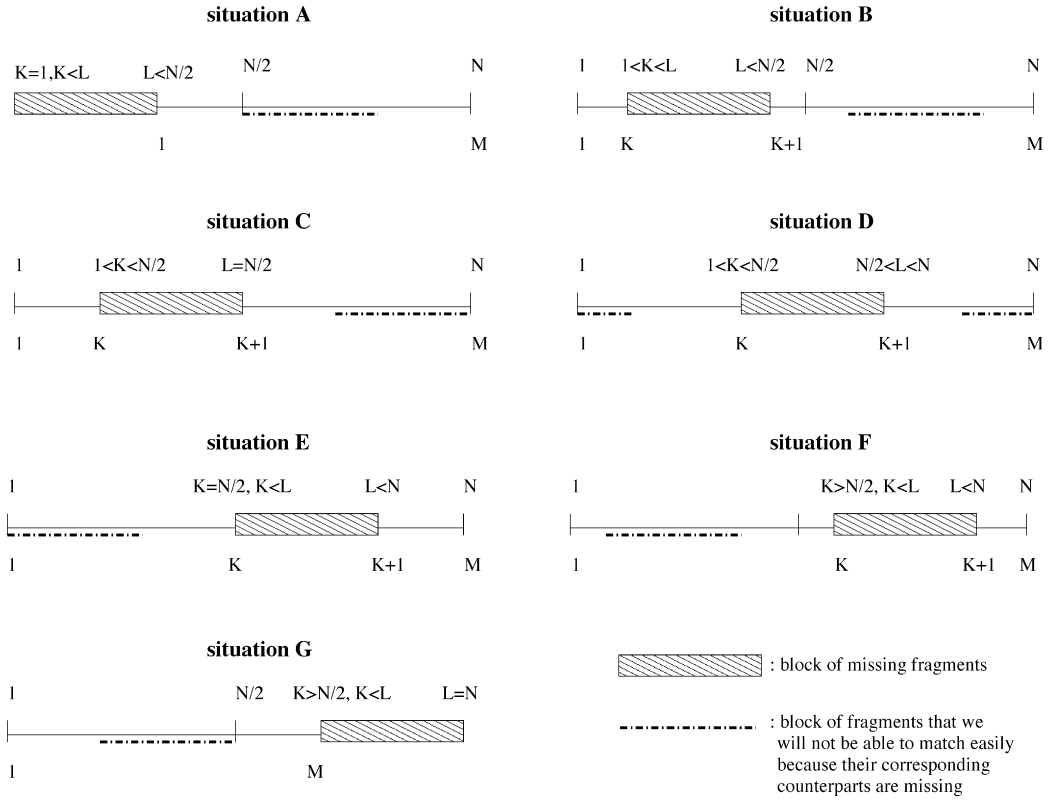

Fig. 7. Missing fragment situations that we will be able to compensate for; the numbers or variables indicated on top of each stack numbering line represent the real fragment numbering, the numbers or variables underneath each stack-line represent the fragment numbers as they will be seen upon recovery (when missing segments are effectively removed).

documents for which a large number of the recovered fragments have been partially damaged. Let us illustrate this as follows. Suppose that the lower half of each of the fragments shown in Fig. 1 would be burned or charcoaled. Clearly, this could complicate any unstructured manual or semi-automated reconstruction process considerably. Using the algorithms proposed in this paper, the reconstruction process would also become more difficult, but as soon as enough information is obtained for confirming the ripping process hypothesis under consideration, the fragment ordering information can again be exploited fully.

We now discuss the main "external" approach for tackling the missing fragment problem. Let us assume that M fragments are recovered. The problem can then be solved by (i) determining how many fragments, i.e., $D = L - K$, have been lost (see Fig. 7) and (ii) determining which situation, see also Fig. 7, corresponds to the recovered fragment stack.

As such, Table 1 can be used to determine the most likely values for the number of missing fragments, i.e., D. To start with, we can eliminate all table entries that have a value smaller than M, because N must be larger than (or equal to) M. Next, we look for the closest table entry T_0, $T_0 \geq M$. We compute $D_0 = T_0 - M$, and determine if this D_0 would be a correct candidate value for $L - K$. If D_0 does not test positive we take the next closest value T_1 from the table, compute D_1, etc. If we arrive at a value $T_i > 2M$ we terminate the process (this can only occur if more than half the number of original fragments has gone missing; we assume that this will not be the case). Alternatively, one can also eliminate any table entries with a value larger than $2M$ before the D_i testing algorithm is started.

Algorithm 3 contains the main missing-block reconstruction procedure we propose.

Algorithm 3. Main algorithm for reconstructing documents when a single consecutive block of fragments is (possibly) missing or severely damaged

```
set i = 0  //simple iteration counter
set SITUATION= unknown
initialize the set of virtual fragments V^(1) to empty
//V^(1) will contain the non-matchable fragments that are to be
//virtually merged with their missing counterparts (see also main text)
set s = not found
//s is used by the sub-procedures to possibly indicate the starting
//position of non-matchable fragments; see dotted lines Fig. 7
DFOUND= false  // did we find a correct D_i ?
eliminate all table entries with a value < M in Table 1
determine T_i in Table 1
(smallest still valid column P, T_i closest (or ==) to M)
//we assume that less than half the number of fragments is missing:
while T_i < 2M  do
   set D_i = T_i − M
   //let us now test D_i; are we missing D_i fragments or not?
   if  S^(0)(1) pairs with S^(0)(1 + T_i/2)  then
      execute procedure Algorithm 4 // testing for situation A, F, or G
   else
      if  remnant S^(0)(1) pairs with S^(0)(1 + T_i/2 − D_i)  then
         execute procedure Algorithm 5 // testing for situation B or C
      else
         execute procedure Algorithm 6 // testing for situation D or E
      end if
   end if
   if DFOUND == true then
      set N = T_i
      report SITUATION and N
      stop; // see main text on how to continue
   else
      delete T_i from Table 1, and set i = i + 1
      determine new T_i in Table 1
(smallest still valid column P, T_i closest to M)
   end if
end while
error; stop; // one or more assumptions violated
```

The only thing that now remains to be explained is how one can test a specific candidate D_i. This ''internal test'' can be accomplished using the subroutine procedures described in Algorithms 4–6; Algorithm 3 will call and execute the required subroutine when needed. Due to a lack of space we can not discuss all these algorithms and situations (see Fig. 7) in detail here. However, for each possible situation, either the test will fail (indicating that another D_i should be picked and evaluated as described above), the first reconstruction iteration will be completed or an error will terminate the procedure (one of the assumptions must be violated). Below we will discuss an example reconstruction procedure. We hope that the reader will be able to understand and verify all other components of the algorithms; the algorithms can be studied most easily by experimenting with fragment stacks similar to the example that is discussed below.

Algorithm 4. Procedure for testing D_i for situation A, F, or G

```
1: for all j = 1, ..., T_i/2 − D_i do
2:    if S^(0)(j) pairs with S^(0)(j + T_i/2) then
3:       form S^(1)(j) = (S^(0)(j), S^(0)(j + T_i/2))
4:    else
5:       go to line 18 // testing for situation F
6:    end if
7: end for
8: set DFOUND=true
9: if D_i != 0 then
10:   label S^(0)(j) as V^(1)(k) for j = T_i/2 − D_i + 1, ..., T_i/2; k = j − T_i/2 + D_i
11:   set SITUATION = A-OR-G (main text explains on how to proceed)
12: else
13:   set SITUATION = A // catch D_i = 0 missing fragments situation
14: end if
15: return to main algorithm
16: // ************************************************************
17: // testing for situation F
18: set s = j
19: for all j = s + D_i, ..., T_i/2 do
20:   if S^(0)(j) pairs with S^(0)(j + T_i/2 − D_i) then
21:      form S^(1)(j) = (S^(0)(j), S^(0)(j + T_i/2 − D_i))
22:   else
23:      if j == s + D_i then
24:         return to main algorithm (DFOUND remains false)
25:      else
26:         error; stop; // one or more assumptions violated
27:      end if
28:   end if
29: end for
30: set DFOUND=true
31: label S^(0)(j) as V^(1)(k) for j = s, ..., s + D_i − 1; k = j − s + 1
32: set SITUATION = F (main text explains on how to proceed)
33: return to main algorithm
```

Algorithm 5. Procedure for testing D_i for situation B or C

```
1: for all j = 1, ..., T_i/2 − D_i do
2:    if S^(0)(j) pairs with S^(0)(j + T_i/2 − D_i) then
3:       form S^(1)(j) = (S^(0)(j), S^(0)(j + T_i/2 − D_i))
4:    else
5:       go to line 14 // testing for situation B
6:    end if
7: end for
8: set DFOUND=true
```

label $S^{(0)}(j)$ as $V^{(1)}(k)$ for $j = T_i - 2\,D_i + 1, \ldots, T_i - D_i$; $k = j - T_i + 2\,D_i$
set SITUATION = C (main text explains on how to proceed)
return to main algorithm
// **
// testing for situation B
set $s = j$
for all $j = s, \ldots, T_i/2 - D_i$ **do**
  if $S^{(0)}(j)$ pairs with $S^{(0)}(j + T_i/2)$ **then**
    form $S^{(1)}(j) = (S^{(0)}(j),\ S^{(0)}(j + T_i/2))$
  else
    if $j == s$ **then**
      return to main algorithm (DFOUND remains false)
    else
      error; stop; // one or more assumptions violated
    end if
  end if
end for
set DFOUND=**true**
label $S^{(0)}(j)$ as $V^{(1)}(k)$ for $j = s + T_i/2 - D_i, \ldots, s + T_i/2 - 1$;
$k = j - s - T_i/2 + D_i + 1$
set SITUATION = B (main text explains on how to proceed)
return to main algorithm

Algorithm 6. Procedure for testing D_i for situation D or E

for all $j = 1, \ldots, D_i$ **do**
  if $S^{(0)}(j)$ pairs with $S^{(0)}(j + T_i/2 - D_i)$ **then**
    go to line 19 // testing for situation D
  end if
end for
for all $j = D_i + 1, \ldots, T_i/2$ **do**
  if $S^{(0)}(j)$ pairs with $S^{(0)}(j + T_i/2 - D_i)$ **then**
    form $S^{(1)}(j) = (S^{(0)}(j),\ S^{(0)}(j + T_i/2 - D_i))$
  else
    error; stop; //one or more assumptions violated
  end if
end for
set DFOUND=**true**
label $S^{(0)}(j)$ as $V^{(1)}(k)$ for $j = 1, \ldots, D_i$; $k = j$
set SITUATION = E (main text explains on how to proceed)
return to main algorithm
// **
// testing for situation D
set $s = j$
for all $j = s, \ldots, T_i/2 - D_i + s - 1$ **do**
  if $S^{(0)}(j)$ pairs with $S^{(0)}(j + T_i/2 - D_i)$ **then**
    form $S^{(1)}(j) = (S^{(0)}(j),\ S^{(0)}(j + T_i/2 - D_i))$
  else
    error; stop; //one or more assumptions violated
  end if
end for
set DFOUND=**true**
label $S^{(0)}(j)$ as $V^{(1)}(k)$ for $j = 1, \ldots, s - 1$; $k = j$
label $S^{(0)}(j)$ as $V^{(1)}(k)$ for $j = T_i - 2D_i + s, \ldots, T_i - D_i$; $k = j$
set SITUATION = D (main text explains on how to proceed)
return to main algorithm

Table 2
The possibilities for D_i that remain to be explored by the missing block algorithm for the example discussed in the main text

R	P					
	1	3	5	7	9	11
1	–	–	–	14	18	22
2	–	–	20	–	–	–
3	–	24	–	–	–	–
4	16	–	–	–	–	–

P: The number of pages, R: the number of ripping iterations.

Next, as can be noted from examining the proposed algorithms it needs to be decided what should be done with the virtual fragments $V^{(1)}$. As indicated in the algorithms these fragments are used to virtually merge, or rather relabel, the non-matchable fragments (dotted lines in Fig. 7). This is important as for some reconstruction situations it may not be immediately clear which situation corresponds to the recovered fragment stack. For example from the reconstruction point of view, the difference between situation A and G can not be determined immediately (see also lines 8–15 in Algorithm 4). For making the correct decision on what to do in these situations one needs to consider the next reconstruction iteration, i.e., the reconstruction iteration used for forming $S^{(2)}$. Note that in case of situation A, the fragments $V^{(1)}$ should be put before the (meta)fragments $S^{(1)}$. For situation G, the set $V^{(1)}$ should be put after $S^{(1)}$. Hence, by choosing either one of these possibilities and determining if a successful match can be found using the standard pairing algorithms (i.e., Algorithms 1 and 2) for $S^{(2)}$, one can determine if the correct ordering for the sets $V^{(1)}$ and $S^{(1)}$ was chosen, and the correct situation can be determined. For single situation solutions (e.g., situation B or C) one simply needs to position the fragments $V^{(1)}$ into their correct position (using Fig. 7) and proceed with the standard pairing algorithms for forming $S^{(2)}$.

Please note that there is an important reason for using $V^{(1)}$, and not just the corresponding $S^{(0)}$ fragments. For manual reconstruction methods we suggest to mark or tag the physical fragments so that that they can be easily identified; matching meta-fragments that contain these virtual fragments may be more difficult as some of their neighborhood context is missing. Hence, the labeling can yield a visual cue for performing more attentive fragment pairing.

Let us now demonstrate the algorithm with an example reconstruction procedure. Suppose we have 16 original fragments, i.e., $S^{(0,\mathrm{org})}$, and three of them (underlined) are lost: `01,02,03,04,05,06,07,08,09,10,11,12,13,14-,15,16`. For this stack the correct first iteration of fragment pairings will correspond to the set: {(01,09), …, (j, $j+8$), …, (08,16)}. The stack that is recovered, i.e., $S^{(0)}(i)$, will be numbered: `01,02,03,04,05,06,07,08,09,10,11, , , ,12,13`. Next, by considering both line 10 (eliminating all entries <13) and line 13 (eliminating all entries ≥ 26) in Algorithm 3, Table 1 can be reduced to Table 2. The algorithm will then select the table entry $P=1, R=4$; yielding $D_0 = 16-13 = 3$. In line 16 in Algorithm 3, $S^{(0)}(1)$ will match with $S^{(0)}(9)$ because $S^{(0)}(1) = S^{(0,\mathrm{org})}(1)$ and $S^{(0)}(9) = S^{(0,org)}(9)$ and `(01,09)` is a correct pair for the original set of fragments. As a result, the procedure described by Algorithm 4 will be executed. Algorithm 4 will form $S^{(1)}(j), j=1,\ldots,3$, using lines 1–7, and $S^{(1)}(j), j=7,\ldots,8$ using lines 19–29. The situation that is identified will correspond to the correct situation (F). Finally, $V(1)(k), k=1,\ldots,3$ will need to be inserted after $S^{(1)}(3)$ and the reconstruction process can then be continued using the reconstruction algorithms described earlier (for non-missing-block situations).

The algorithms proposed in this section could be integrated more closely, but we decided not to do this as we feel that, in combination with Fig. 7, it is easier to solve and understand each subproblem separately.

Determining a theoretical estimate for the speed-up that can be obtained for missing fragment problems is not as simple as it was in the previous sections; e.g., the reconstruction process is determined by the number of iterations i needed for finding a correct D_i. In other words, if, e.g., a large number of fragments has been lost, a number of D_i trial-and-error matching tests may need to be performed.

Despite the fact that some D_i tests and fragment matches may fail, the proposed algorithms still yield a well-structured and formal approach that, on average, should certainly remain more efficient than performing random search procedures. Additionally, some of our future work will focus on the adaptation of the proposed algorithms for implementing a computer program with which the user can interact. The most important novelty that we are currently investigating is the extension of this program in order for it to allow one to assist the algorithmic reconstruction process by providing information about visually discovered matches at any time during reconstruction process; *quickly* spotted random matches may help speed-up the determination of how many fragments went missing, which missing-block situation corresponds to the recovered stack, etc.

10. Conclusion

In this paper, we proposed a formal analysis method that can be used for devising efficient reassembly procedures when a set of document remnants can be recovered as an ordered stack of fragments.

For small scale problems that can also be solved using any homespun or "random search" method, the proposed approaches offer a better understanding of the underlying theoretical ripping and reconstruction problems.

Whenever a larger number of fragments or document pages needs to be recomposed, the algorithms discussed in this paper can be used to complete a reconstruction task using a more structured methodology that also requires considerably less fragment matching operations. As a result of this, this paper offers very fast strategies for both manual and (semi-)automated reconstruction tasks.

The proposed methods are currently being studied for integration with our earlier work on semi-automatic reconstruction techniques [10–12]. Additionally, we are extending the algorithms in order to be able to compensate for substack split-ups. This can happen when, e.g., a single ripped-up stack

is thrown into a bin; some subsets of the fragments may stick to each other, while others may not. As a result of this, when these multiple substacks are independently recovered, their relative ordering may often be unknown. Preliminary results indicate that their relative ordering can often still be deduced quickly by performing an intelligent sequence of matching experiments.

As already pointed out above, compensating for more complicated situations of fragment loss or fragment displacement that would occur during the ripping or recovery process, may also be an interesting topic for future research. A similar remark holds for irregular ripping patterns, tear-of ripping for large fragments that may have certain parts sticking out of the stack, etc.

We hope that this paper may inspire other researchers to initiate or contribute some of their own insights, real-life casework experience, or (future) research.

Acknowledgements

The author wishes to thank the European Network of Forensic Science Institutes (ENFSI) committees and working groups and the National Bureau of Investigation in Finland for organizing the EAFS 2006 conference. The discussion of his work during the corresponding poster session at the EAFS 2006 meeting motivated the author to compile and submit this paper.

References

[1] P. De Smet, Fragment stack analysis techniques for efficient reconstruction of ripped-up documents, in: Proc. 57th Annual Meeting American Academy of Forensic Sciences (AAFS2005), 2005, 347–348.

[2] P. De Smet, Reconstruction of ripped-up documents based on the analysis of recovered stacks of fragments, in: Proc. 4th European Academy of Forensic Science Conference (EAFS2006), 2006, 151–152.

[3] H. Bunke, G. Kaufmann, Jigsaw puzzle solving using approximate string matching and best-first search, in: D. Chetverikov, W.G. Kropatsch (Eds.), Proc. 5th Int. Conference Computer Analysis of Images and Patterns (CAIP'93), Lecture Notes in Computer Science, vol. 719, Springer, 1993, pp. 477–484.

[4] D. Goldberg, C. Malon, M. Bern, A global approach to automatic solution of jigsaw puzzles, Comput. Geom. Theory Appl. 28 (2–3) (2004) 165–174.

[5] F.-H. Yao, G.-F. Shao, A shape and image merging technique to solve jigsaw puzzles, Pattern Recognit. Lett. 24 (12) (2003) 1819–1835.

[6] J. De Bock, P. De Smet, W. Philips, J. D'Haeyer, Constructing the Topological solution of jigsaw puzzles, in: IEEE Int. Conference on Image Processing (ICIP 2004), vol. 3, 2004, 2127–2130.

[7] H. Leitao, J. Stolfi, A multi-scale method for the reassembly of two-dimensional fragmented objects, IEEE Trans. Pattern Anal. Machine Intell. 24 (2002) 1239–1251.

[8] E. Justino, L.S. Oliveira, C. Freitas, Reconstructing shredded documents through feature matching, Forensic Sci. Int. 160 (2006) 140–147.

[9] C. Papaodysseus, T. Panagopoulos, M. Exarhos, C. Triantafillou, D. Fragoulis, C. Doumas, Contour-shape based reconstruction of fragmented, 1600 BC wallpaintings, IEEE Trans. Signal Process. 50 (6) (2002) 1277–1288.

[10] P. De Smet, J. De Bock, E. Corluy, Semi-automatic jigsaw puzzle reconstruction of fragmented documents, in: Proc. 3rd Triennial Meeting of the European Academy of Forensic Science, 2003, pp. 29–30 (EAFS2003, also published in: Forensic Science International 136 (1:suppl.1)).

[11] P. De Smet, J. De Bock, E. Corluy, Computer vision techniques for semi-automatic reconstruction of ripped-up documents, in: SPIE AeroSense Conference Proc. 5108B Investigative Image Processing, 2003, 189–197.

[12] J. De Bock, P. De Smet, Semi-automatic reconstruction of fragmented 2D objects, in: Proc. 56th Annual Meeting American Academy of Forensic Sciences (AAFS2004), 2004, pp. 147–148 (see also http://telin.ugent.be/ipi/foco/).

[13] P. De Smet, J. De Bock, W. Philips, Semi automatic reconstruction of strip shredded documents, in: Proc. SPIE Electronic Imaging, Image and Video Communications and Processing, vol. 5685, Investigative Image Processing III (EI123), Society of Photo-Optical Instrumentation Engineers (SPIE), Bellingham, WA, USA, 2005, pp. 239–248